Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Комп’ютерний практикум №5

Технології паралельних обчислень

Тема: Застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності

Виконав
студент групи ІП-11:
Панченко С. В.

Перевірила:
Стеценко І.В.

Київ 2024

ЗМІСТ

1 Завдання.....6

2 Виконання.....7

2.1 Структура проєкту.....7

2.2 Робота системи масового обслуговування.....7

3 Висновок.....8

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ.....9

1 ЗАВДАННЯ

1) З використанням пулу потоків побудувати алгоритм імітації багатоканальної системи масового обслуговування з обмеженою чергою, відтворюючи функціонування кожного каналу обслуговування в окремій підзадачі. Результатом виконання алгоритму є розраховані значення середньої довжини черги та ймовірності відмови. 40 балів.
2) З використанням багатопоточної технології організувати паралельне виконання прогонів імітаційної моделі СМО для отримання статистично значимої оцінки середньої довжини черги та ймовірності відмови. 20 балів.
3) Виводити результати імітаційного моделювання (стан моделі та чисельні значення вихідних змінних) в окремому потоці для динамічного відтворення імітації системи. 20 балів.
4) Побудувати теоретичні оцінки показників ефективності для одного з алгоритмів практичних завдань 2-5. 20 балів.

2 ВИКОНАННЯ

2.1 Структура проєкту

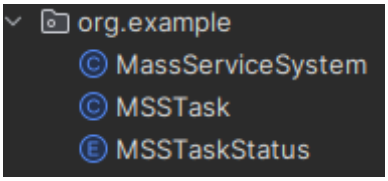

Рисунок 2.1.1 - Структура проєкту

MassServiceSystem — клас, що репрезентує собою систему масового обслуговування, що під собою включає чергу з MSSTask, логер та екзек'ютор. При створенні об'єкта класу у констркторі створюється окремий потік, який логує інформацію про те, що відбувається всередині системи.

MSSTask — клас, що являє собою окреме завдання та імплементує інтерфейс Runnable.

MSSTaskStatus — енам, що включає в себе можливі статуси, які приймає MSSTask.

2.2 Робота системи масового обслуговування

Під час роботи системи вона логує середню довжину черги та шанс відмови від початку роботи до поточного моменту.

```
14:53:21.530 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.802528]
14:53:21.530 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385843]
14:53:21.533 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.800126]
14:53:21.533 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385737]
14:53:21.535 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.797728]
14:53:21.536 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385685]
14:53:21.537 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.794136]
14:53:21.538 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385646]
14:53:21.540 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.791745]
14:53:21.540 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385619]
14:53:21.542 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.788165]
14:53:21.542 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385606]
14:53:21.543 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.785782]
14:53:21.544 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385580]
14:53:21.546 [Thread-0] INFO  MassServiceSystem - [MEAN QUEUE LENGTH: 3.783401]
14:53:21.547 [Thread-0] INFO  MassServiceSystem - [CANCELLATION PROBABILITY: 0.385475]
```

Рисунок 2.2.1 - Логування середньої довжини черги та шансу відмови від початку роботи до поточного моменту

3 ВИСНОВОК

Під час лабораторної роботи опрацювали застосування високорівневих засобів паралельного програмування для побудови алгоритмів імітації та дослідження їх ефективності.

У результаті отримали систему масового обслуговування, що у режимі реального часу в окремому потоці описує середню довжину черги та ймовірність відмови вхідного завдання.

ДОДАТОК А ТЕКСТИ ПРОГРАМНОГО КОДУ

Тексти програмного коду

(Найменування програми (документа))

Жорсткий диск

(Вид носія даних)

(Обсяг програми (документа), арк.)

Студента групи ІП-113 курсу

Панченка С. В

```
// ./Lab5/Lab5/src/main/java/org/example/MSSTask.java

package org.example;

public class MSSTask implements Runnable {
    private static final String MAX_LESS_THAN_MIN = "maxSleepTime < minSleepTime";
    private final long minSleepTime;
    private final long maxSleepTime;

    private final Object statusLock = new Object();
    private MSSTaskStatus status = MSSTaskStatus.Waiting;

    public MSSTask(int minSleepTime, int maxSleepTime) {
        if(minSleepTime > maxSleepTime) {
            throw new IllegalArgumentException(MAX_LESS_THAN_MIN);
        }
        this.minSleepTime = minSleepTime;
        this.maxSleepTime = maxSleepTime;
    }

    @Override
    public void run() {
        setStatus(MSSTaskStatus.Running);

        try {
            Thread.sleep(genSleepTime());
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
            setStatus(MSSTaskStatus.Finished);
        }

        private long genSleepTime() {
            return (long) (Math.random() * (maxSleepTime - minSleepTime) +
minSleepTime);
        }

        private void setStatus(MSSTaskStatus status) {
            synchronized(statusLock) {
                this.status = status;
            }
        }

        public MSSTaskStatus getStatus() {
            synchronized(statusLock) {
                return status;
            }
        }

        public void cancel() {
            setStatus(MSSTaskStatus.Cancelled);
        }

    }


    // ./Lab5/Lab5/src/main/java/org/example/MSSTaskStatus.java

    package org.example;

    public enum MSSTaskStatus {
```

```
        Cancelled,
        Waiting,
        Running,
        Finished
    }


    // ./Lab5/Lab5/src/main/java/org/example/MassServiceSystem.java

    package org.example;

    import java.util.ArrayList;
    import java.util.concurrent.ExecutorService;
    import java.util.concurrent.Executors;
    import java.util.concurrent.atomic.AtomicBoolean;
    import java.util.stream.Stream;

    import org.apache.logging.log4j.LogManager;
    import org.apache.logging.log4j.Logger;

    public class MassServiceSystem {
        private static final String SERVICE_ALREADY_STOPPED = "Service
already stopped";
        private static final String MEAN_QUEUE_LENGTH = "[MEAN
QUEUE LENGTH: %f]";
        private static final String CANCELLATION_PROBABILITY =
"[CANCELLATION PROBABILITY: %f]";

        private static final int LOG_DELAY_MILLIS = 2;

        private final int maxWaitedTasks;
        private final int maxRunningTasks;
```

```
        private final Logger logger;

        private final ExecutorService executor;

        private final ArrayList<MSSTask> tasks = new ArrayList<>();

        private final AtomicBoolean isRunning = new AtomicBoolean(true);

        private final long startTime = System.currentTimeMillis();
        private long prevTime = System.currentTimeMillis();
        private long currentTime = System.currentTimeMillis();

        private double queueTimeSum = 0;
        private double cancellationTimeSum = 0;

        public static void main(String[] args) {
            var minSleepTime = 1;
            var maxSleepTime = 5;
            var totalTasks = 10000000;
            var maximumRunningTasks = 10;
            var maximumWaitedTasks = 10;
            var logger = LogManager.getLogger("MassServiceSystem");
            var mss = new MassServiceSystem(maximumRunningTasks,
maximumWaitedTasks, logger);
            for(var i = 0; i < totalTasks; ++i) {
                mss.addTask(new MSSTask(minSleepTime, maxSleepTime));
            }
        }

        public MassServiceSystem(int maxRunningTasks, int maxWaitedTasks,
Logger logger) {
```

```
            this.executor = Executors.newFixedThreadPool(maxRunningTasks);
            this.maxWaitedTasks = maxWaitedTasks;
            this.maxRunningTasks = maxRunningTasks;
            this.logger = logger;
            var logThread = new Thread(this::log);
            logThread.start();
        }

        public void addTask(MSSTask task) {
            var sizes = getStatusSizes(MSSTaskStatus.Running,
MSSTaskStatus.Waiting);;
            synchronized(tasks) {
                tasks.add(task);
            }
            synchronized(executor) {
                if(sizes[0] < maxRunningTasks || sizes[1] < maxWaitedTasks)
{
                    executor.execute(task);
                    return;
                }
            }
            task.cancel();
        }
        private void log() {
            while(isRunning.get()) {
                currentTime = System.currentTimeMillis();

                if(currentTime - prevTime < LOG_DELAY_MILLIS) {
                    continue;
                }

                var sizes = getStatusSizes(MSSTaskStatus.Waiting,
```

```
MSSTaskStatus.Cancelled);

                    queueTimeSum += sizes[0] * (currentTime - prevTime);
                    var meanWaiting = queueTimeSum / (currentTime -
startTime);

    logger.info(MEAN_QUEUE_LENGTH.formatted(meanWaiting));

                    cancellationTimeSum += sizes[1] * (currentTime - prevTime);
                    var meanCancellation = cancellationTimeSum / (currentTime -
startTime);

                    var prob = meanCancellation / getTasksSize();

    logger.info(CANCELLATION_PROBABILITY.formatted(prob));

                    prevTime = currentTime;
                }
                synchronized(executor) {
                    executor.shutdown();
                }
            }

            private long[] getStatusSizes(MSSTaskStatus... statuses) {
                synchronized(tasks) {
                    return Stream.of(statuses).mapToLong(
                        s -> tasks.stream().filter(t -> t.getStatus() == s).count()
                    ).toArray();
                }
            }

            private int getTasksSize() {
```

```
            synchronized(tasks) {
                return tasks.size();
            }
        }

        public void shutdown() {
            if(!isRunning.get()) {
                throw new IllegalStateException(SERVICE_ALREADY_STOPPED);
            }
            isRunning.set(false);
        }
    }
```